Dina Research School:

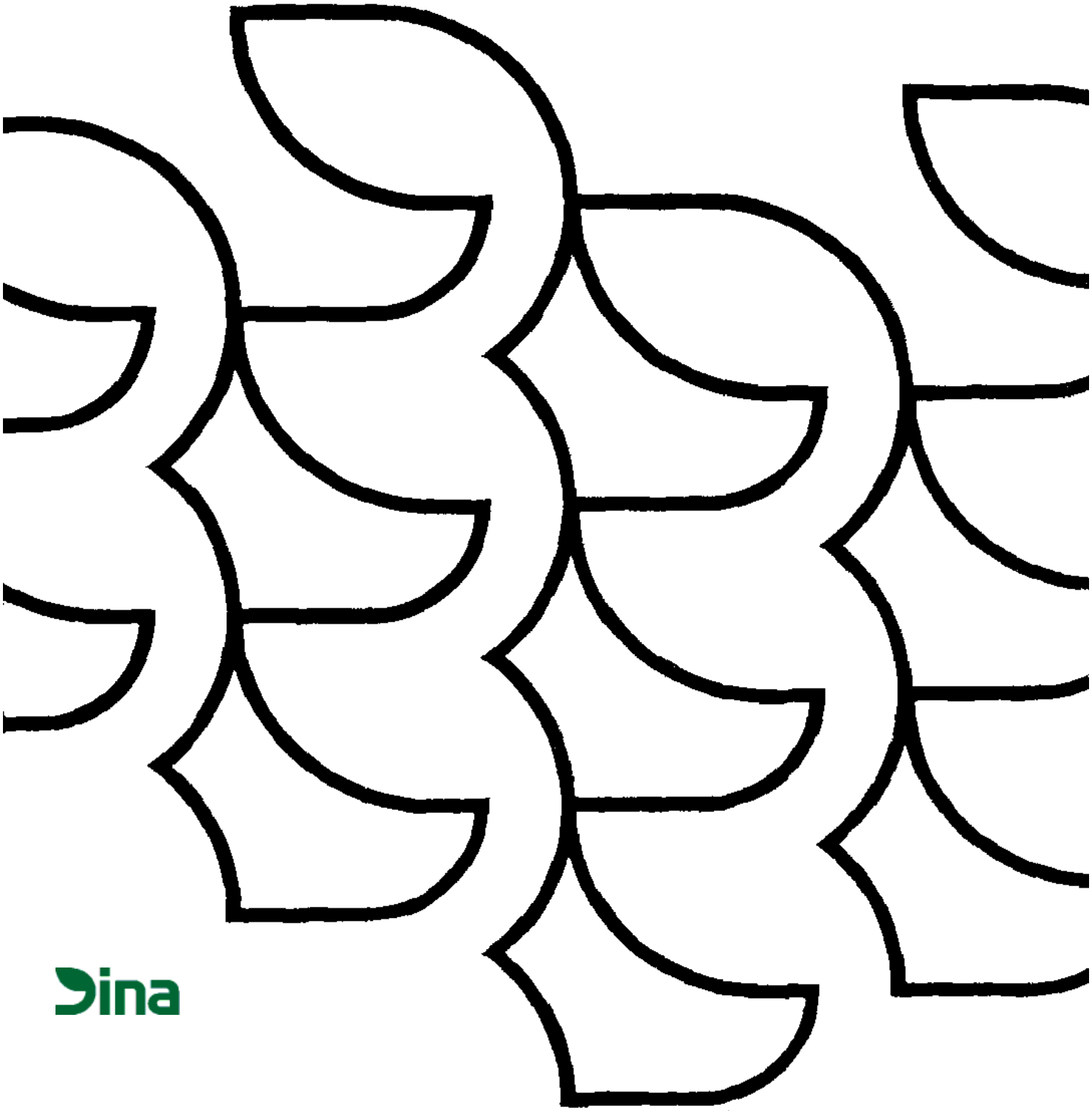
Workshop October 30-31, 2003

Monte Carlo methods for Hierarchical (Mixed) Models

Koldkærgaard Landboskole, Skejby.

Edited by Erik Jørgensen & Rasmus Waagepetersen

Dina Notat No. 107 · October 2003



Dina Research School:

Workshop October 30-31, 2003

Koldkærgaard Landboskole, Skejby.

Edited by Erik Jørgensen & Rasmus Waagepetersen

Dina Notat No. 107 · October 2003

This note is also available on www at URL:
http://www.dina.dk/phd/w/w12/notat107.pdf

Dina Research School
Department of Animal Science and Animal Health
Royal Veterinary and Agricultural University
Grønnegårdsvej 8, DK-1870
Frederiksberg C

Contents

Introduction

Studies within agricultural research are often characterised by an hierarchical design. Examples are studies based on measurements on plants within fields within farms, or similarly animals within herds within breeds. Hierarchical design arise both in controlled experiments and in epidemiological studies. Statistical analysis of these data are complicated because the design leads to dependency between observations. These dependencies need to be handled in the statistical model. Therefore, parameter estimation becomes very complex in most cases, and specialised numerical methods are necessary. Monte Carlo methods are becoming increasingly popular for this purpose. The workshop will present the principles behind the methods, and their application for analysing data from the hierarchical designs will be demonstrated. Typically, the models include both *fixed* effects (e.g., treatment) and *random* components that models the dependency between observations (e.g., a farm effect). Therefore, such models are often called mixed models.

The lectures and case studies will be accompanied by computer exercises where the workshop participants can get a hands on experience with the methodology.

The theoretical basis is likelihood based inference for these hierarchical or mixed models, and topics will include both traditional models and so called latent class or mixture models. Different Monte Carlo methods will be presented, e.g. importance sampling and Markov Chain Monte Carlo (MCMC). Models based on both normal (Gaussian) and other probability distributions (generalised linear mixed models) will be covered.

The workshop is aiming at

- Ph.D.-students in the agricultural and biological sciences, whose work involves data from hierarchical designs and who are interested in learning about the new possibilities for analysing these data.
- Ph.D.-students with a background in mathematics, computer science, or engineering who are interested in learning more about the applications in the biological sciences.

1 Program

Thursday, October 30

11.00 Arrival and accommodation.
12.00 Lunch

13.00 **Introduction and presentation of participants**
Erik Jørgensen, Dina Research School.

13.15 **Theory session I:**
Rasmus Waagepetersen, Aalborg University.
The concepts of hierarchical and generalized linear mixed models are discussed in relation to various datasets from the agricultural sciences. A basic problem with these models is that the likelihood function is not available in closed form. We consider various numerical techniques for calculating the likelihood function with a particular focus on Monte Carlo methods.

14.00 **Short break followed by computer exercises.**

14.45 **Discussion of exercises**
15.00 Coffee break.

15.30 **Theory session II:**
Rasmus Waagepetersen, Aalborg University.
We continue the discussion of Monte Carlo methods and consider importance sampling and Markov chain Monte Carlo methods for generating samples from an importance sampling distribution.

16.15 **Computer exercises.**

17.00 **Theory session III:**
Rasmus Waagepetersen, Aalborg University.
We discuss the Bayesian approach to inference for hierarchical models and how it can be implemented using the software BUGS.

18.00 Dinner.

19.00 **Case study I: Estimation of sero-prevalence of paratuberculosis with mixture models including covariates .**
Nils Toft, Animal Science and Animal Health, The Royal Veterinary and Agricultural University
A mixture model was developed to estimate strata-specific sero-prevalences of paratuberculosis among cows in 100 Danish dairy herds using milk samples taken in three sampling rounds. Estimates showed that among 1st parity cows in round three, 6

19.45 **Computer exercises (continued)**.

21.45 Coffee and sandwich.

Friday, October 31

7.30 Breakfast.

8.30 **Discussion of computer exercises**.

9.00 **Case II: Spatial GLMM**
Ole Fredslund Christensen , Center for Bioinformatics, Aarhus Universitet .

9.45 Coffee Break.

10.00 **Theory IV:**.
Rasmus Waagepetersen, Aalborg University.
The final lecture is devoted to more specialized topics in MCMC and hierarchical models with non-Gaussian random effects.

10.30 Break.

10.35 Computer exercises.

11.15 **Case III: Preference Experiment with Repeated Measurement**
Erik Jørgensen
Preference for different rooting materials of pigs is studied using a choice experiment setup, where each pig repeatedly is given the choice between three different materials

11.45 **Discussion and Closing**.
Erik Jørgensen, Dina Research School.

12.00 Lunch and departure.

2 List of participants

Monte Carlo methods for Hierarchical (Mixed) Models.

Koldkærgård Landboskole, Skejby, October 30-31 2003.

List of Participants

Name	Email	Phone
Saeid Ansari-Mahyari	SaeidAnsari.Mahyari@agrsci.dk	+45 8999 1331
Karsten Dalsgaard Bjerre	kdb@kvl.dk	+45 3528 2641
Adrian Marc Bolliger	amb@kvl.dk	+45 3528 3497
Michael Brogaard	Michael.Brogaard@agrsci.dk	+45 58 34 42
Lars Holm Damgaard	Lars.Damgaard@agrsci.dk	
Omar A. N. Daraghmeh	oand@kvl.dk	+45 3528 3544
Niels Halberg	niels.halberg@agrsci.dk	
Erik Jørgensen	Erik.Jorgensen@agrsci.dk	89 99 19 00, dir: 89 99 12 30
Thomas Larsen	thl@DMU.dk	+45 8920 1572
Anders Højlund Nielsen	AndersH.Nielsen@agrsci.dk	+45 89 99 12 02
Elise Norberg	Elise.Norberg@agrsci.dk	+4589991339
Louise Dybdahl Pedersen	Louise.DybdahlPedersen@agrsci.dk	+45 89 99 13 31
Niels Peter Bådsgård	NielsP.Baadsgaard@agrsci.dk	89 99 13 33
Mohammad Mahdi Shariati	Mohammad.Shariati@agrsci.dk	+45 8999 1542
Anders Christian Sørensen	AndersC.Sorensen@agrsci.dk	+4589991224
Nils Toft	nt@dina.kvl.dk	35 28 30 24
Håkan Vigre	hvi@vetints.dk	+45 35 30 02 94
Rasmus Waagepetersen	rw@math.auc.dk	96 35 88 76
Søren Østergaard	Soren.Ostergaard@agrsci.dk	

Author: webmaster.

A quick introduction to Markov chains and Markov chain Monte Carlo (revised version)

Rasmus Waagepetersen
Institute of Mathematical Sciences
Aalborg University

1 Introduction

These notes are intended to provide the reader with knowledge of basic concepts of Markov chain Monte Carlo (MCMC) and hopefully also some intuition about how MCMC works. For more thorough accounts of MCMC the reader is referred to e.g. Gilks *et al.* (1996), Gamerman (1997), or Robert and Casella (1999).

Suppose that we are interested in generating samples from a target probability distribution π on $\mathbb{R}^n$ and that π is so complex that we can not use direct methods for simulation. Using Markov chain Monte Carlo methods it is, however, often feasible to generate an ergodic Markov chain $X_1, X_2, \ldots$ which has π as equilibrium distribution, i.e. after a suitable burn-in period m, $X_{m+1}, X_{m+2}, \ldots$ provides a (correlated) sample from π which can be used e.g. for Monte Carlo computations.

Before we turn to MCMC methods we briefly consider in the next section the concepts of Markov chains and equilibrium distributions.

2 Examples of Markov chains and convergence to a stationary distribution

A Markov chain X is a sequence $X_0, X_1, X_2, \ldots$ of stochastic variables (in short notation $X = (X_l)_{l \geq 0}$) which for all $n > 0$ and all events $A_0, A_1, \ldots, A_n$

satisfies the following conditional independence property:

$$P(X_n \in A_n | X_{n-1} \in A_{n-1}, X_{n-2} \in A_{n-2}, \ldots, X_0 \in A_0) = P(X_n \in A_n | X_{n-1} \in A_{n-1}). \quad (1)$$

That is, the value of the nth variable depends on the past variables only through the immediate predecessor one timestep ahead. The variable X_l, $l \geq 0$ could e.g. designate the average temperature in Denmark on the lth day in 1998, and A_l could be the event that X_l is greater than a temperature T, so that $A = \{x \in \mathbb{R} | x \geq T\}$.

An independent sequence is one example of a Markov chain since in this case

$$P(X_n \in A_n | X_{n-1} \in A_{n-1}, X_{n-2} \in A_{n-2}, \ldots, X_0 \in A_0) = P(X_n \in A_n) \quad (2)$$

which does not depend on the past at all, due to the independence.

As a first specific example we consider a Markov chain on the discrete state space $E = \{0, 1\}$.

Example 1 A Markov chain X on $E = \{0, 1\}$ is determined by the initial distribution given by

$$p_o = P(X_0 = 0) \text{ and } p_1 = P(X_0 = 1),$$

and the one-step transition probabilities given by

$$p_{00} = P(X_{n+1} = 0 | X_n = 0),\ p_{10} = P(X_{n+1} = 0 | X_n = 1),$$
$$p_{01} = 1 - p_{00} \text{ and } p_{11} = 1 - p_{10}.$$

Often the one-step transition probabilities are gathered in a matrix

$$P = \begin{bmatrix} p_{00} & p_{10} \\ p_{10} & p_{11} \end{bmatrix}.$$

□

The next example is a Markov chain with the continuous state space $E = \mathbb{R}$, the real numbers.

Example 2 In this example we consider an autoregressive Markov chain of order 1 (an AR(1)). The chain is given by

a) $X_0 = \mu_0$ (a fixed value).

b) $X_l = \beta X_{l-1} + \epsilon_l$, $l \geq 1$,

where $(\epsilon_l)_{l\geq 1}$ is a sequence of independent and normally distributed "innovations" with $\epsilon \sim N(0, \sigma^2)$. That is, $X_1 = \beta X_0 + \epsilon_1$, $X_2 = \beta X_1 + \epsilon_2$ etc. It is clear that $(X_l)_{l\geq 0}$ forms a Markov chain, since in order to compute X_n the only knowledge required about the past is the value of the predecessor X_{n-1}.

It is easy to calculate the distribution of the variables X_l, $l \geq 0$. They are given as sums of the normal variables ϵ_l:

$$X_n = \beta X_{n-1} + \epsilon_n = \beta^2 X_{n-2} + \beta\epsilon_{n-1} + \epsilon_n = \cdots = \\ \beta^n \mu_0 + \beta^{n-1}\epsilon_1 + \cdots + \beta\epsilon_{n-1} + \epsilon_n = \beta^n \mu_0 + \sum_{l=1}^{n} \beta^{n-l}\epsilon_l, \quad (3)$$

and are therefore normal themselves. It thus only remains to calculate the means and variances:

$$EX_n = \beta^n \mu_0 + \sum_{l=1}^{n} \beta^{n-l} E\epsilon_l = \beta^n \mu_0 \quad (4)$$

$$VarX_n = \sum_{l=1}^{n} (\beta^{n-l})^2 Var\epsilon_l = \sum_{l=1}^{n} (\beta^2)^{n-l}\sigma^2 = \sigma^2 \frac{(\beta^2)^n - 1}{\beta^2 - 1}. \quad (5)$$

(it is here required that $\beta^2 \neq 1$). For the calculation of the variance the formula $\sum_{l=0}^{n} z^l = (z^{n+1} - 1)/(z - 1)$, $z \neq 1$ was used.

From these expressions we see that an AR(1) behaves very differently depending on whether $-1 < \beta < 1$ or $|\beta| > 1$. If $|\beta| < 1$ then EX_n tends to 0 and $Var(X_n)$ tends to $\sigma^2/(1 - \beta^2)$ as n tends to infinity. For large n, X_n thus approaches an $N(0, \sigma^2/(1 - \beta^2))$ distribution. If on the other hand $\beta > 1$, then both EX_n and $Var(X_n)$ tends to infinity as n tends to infinity. This behaviour is reflected by the simulations in Figure 1.

For $-1 < \beta < 1$ the normal distribution $N(0, \sigma^2/(1 - \beta^2))$ is a socalled *invariant* or *stationary* distribution. That is, if X_{n-1} is $N(0, \sigma^2/(1 - \beta^2))$ then this implies that also X_n is normal distributed with mean 0 and variance $\sigma^2/(1 - \beta^2)$. This is seen as follows:

$$E(X_n) = \beta E(X_{n-1}) = 0 = E(X_n),$$

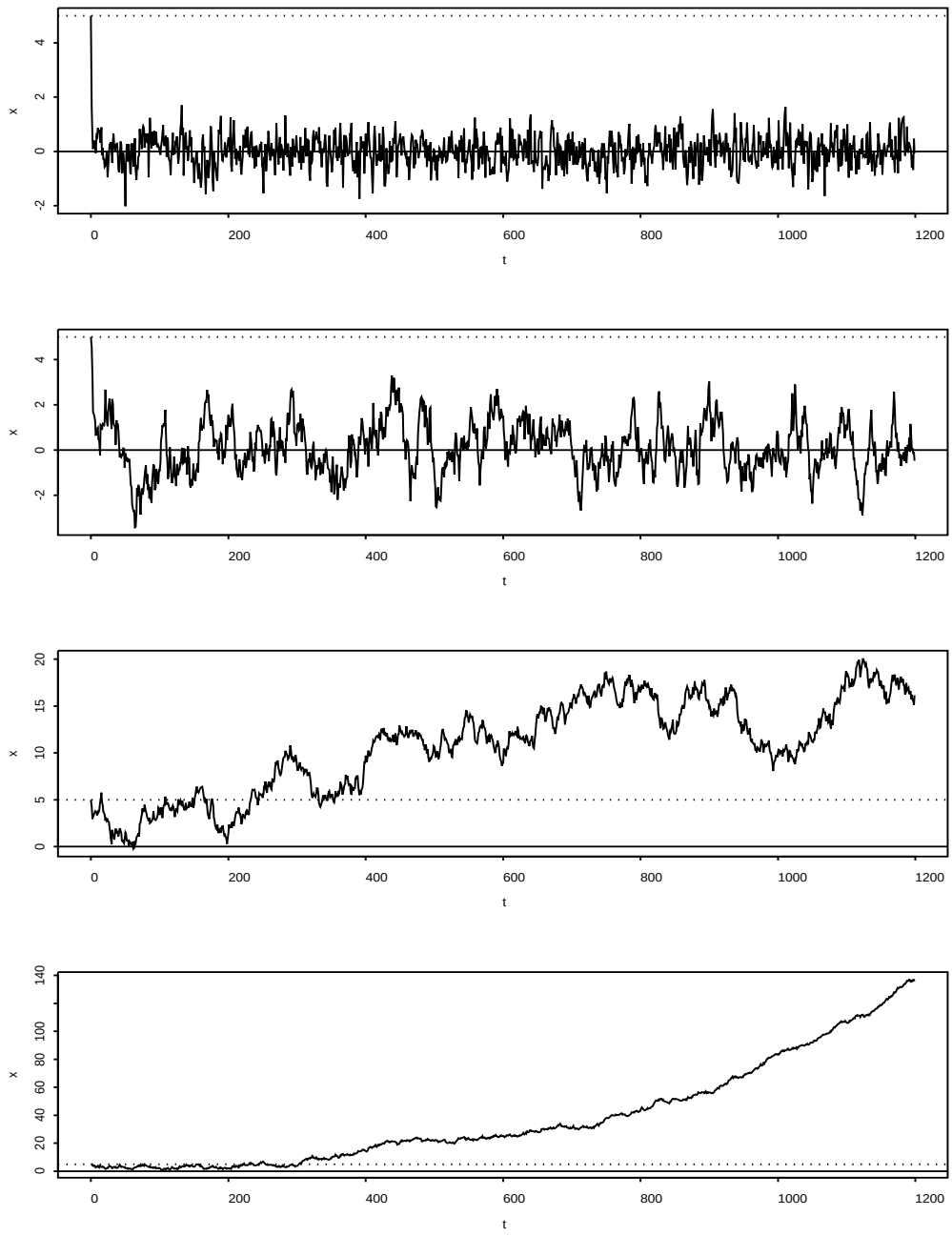


Figure 1: Simulations of AR(1)-chains. For all simulations, $\mu_0 = 5$ and $\sigma^2 = 0.25$. The values of β are, from top to bottom, 0.4, 0.9, 1.0, and 1.0025.

and

$$Var(X_n) = \beta^2 Var(X_{n-1}) + Var(\epsilon_n) = \frac{\beta^2\sigma^2}{1-\beta^2} + \sigma^2 = \\ \sigma^2(\frac{\beta^2}{1-\beta^2} + \frac{1-\beta^2}{1-\beta^2}) = \sigma^2\frac{1}{1-\beta^2} = Var(X_{n-1}).$$

An AR(1) with $|\beta| < 1$ thus illustrates the important properties of "lack of memory" and stochastic stability since the chain converges to an uniquely determined stationary distribution, regardless of the chosen initial value. □

The idea of MCMC is that for an arbitrary distribution π of interest, one can generate a Markov chain whose invariant distribution is given by π, which have the "lack of memory" property as for the AR(1) in Example 2, and which converges to the invariant distribution, so that samples of π can be obtained from the Markov chain.

3 Some theoretical concepts for Markov chains

Let $X = (X_l)_{l\geq 0}$ denote a Markov chain with state space E and let π denote a probability distribution on E. Then

- π is an *invariant distribution* for X if

$$X_l \sim \pi \Rightarrow X_{l+1} \sim \pi,\ l \geq 0$$

- X is *aperiodic* if there does not exist a disjoint subdivision of E into subsets $A_0, \ldots, A_{d-1},\ d \geq 2$, such that

$$P(X_l \in A_{(k+1) \bmod d} | X_{l-1} \in A_k) = 1,\ k = 0, \ldots, d-1,\ l \geq 1$$

- X is *irreducible* if for all $x \in E$ and all events A with $\pi(A) > 0$ there exist an $n \geq 1$ such that $P(X_n \in A | X_0 = x) > 0$.

- X is Harris recurrent if

$$P(\exists n : X_n \in A \mid X_0 = x) = 1$$

for all $x \in E$ and all events A with $\pi(A) > 0$.

Example 1 continued Suppose that π is a distribution on $\{0, 1\}$ given by probabilities π_0 and π_1. Suppose that $P(X_{n-1} = 0) = \pi_0$. Then π is an invariant distribution for the Markov chain given by the one-step transition probabilities p_{00} and p_{10} provided $P(X_{n-1} = 0) = \pi_0$ implies $P(X_n = 0) = \pi_0$, i.e.

$$\pi_0 = P(X_n = 0) = P(X_n = 0|X_{n-1} = 0)\pi_0 + P(X_n = 0|X_{n-1} = 1)\pi_1 = p_{00}\pi_0 + p_{10}\pi_1 \Leftrightarrow \pi_0 = \frac{p_{10}}{p_{10} + p_{01}}.$$

In matrix form the condition is

$$\pi = P\pi.$$

where $\pi = \binom{\pi_0}{\pi_1}$. The Markov chain is periodic if $p_{01} = p_{10} = 1$. □

Example 2 cntd. Let $\pi = N(0, \sigma^2/(1 - \beta^2))$. An AR(1) is then π-irreducible: if $X_0 = x$ then $X_1 \sim N(\beta x, \sigma^2)$ so that $P(X_1 \in A|X_0 = x) > 0$ for any event A with $\pi(A) > 0$. It is also aperiodic: assume first that the Markov chain is periodic and that $X_n = x \in A_k$ (where A_k is one of the subsets in the "periodic" splitting of E above). But $P(X_n \in A_k|X_{n-1} \in A_{k-1}) = 1 > 0$ implies $P(X_{n+1} \in A_k|X_n = x) > 0$ so that $P(X_{n+1} \in A_{k+1 \bmod d}|X_n \in A_k) < 1$. □

3.1 Convergence towards a stationary distribution

Suppose now that π is an invariant distribution for X, and that X is π-irreducible and aperiodic. Then under the further assumption of Harris recurrency, X_n converges in distribution to π for any chosen starting condition for X_0. This means, that for any event A,

$$P(X_n \in A) \to \pi(A),$$

so that X_n can be considered a simulation from the distribution π for large n. Harris recurrence holds under mild conditions for the Metropolis-Hastings samplers described in section 4, provided that irreducibility is fulfilled, see Tierney (1994) and Chan and Geyer (1994).

It is intuitively clear that irreducibility is required for convergence to the stationary distribution, since the chain must be able to reach all parts A of

the state space for which $\pi(A) > 0$. If the chain is periodic and started in A_0, say, then $P(X_n \in A_k) = 1$ whenever $n = 0, k, 2k, \ldots$, and zero otherwise, so that the distribution of X_n can never converge to π.

Convergence to a stationary distribution can e.g. be observed in the two upper plots in Figure 1.

3.2 Convergence of Markov chain Monte Carlo estimates

Let g be a function on E where we wish to estimate $E(g(Z))$, where $Z \sim \pi$. If X is Harris recurrent (irreducible) with stationary distribution π then the empirical average $\sum_{l=0}^{n} g(X_l)/n$ converges:

$$\frac{1}{n}\sum_{l=0}^{n} g(X_l) \to E(g(Z)), \text{ with probability one,}$$

as n tends to infinity, regardless of the chosen initial value for X_0. Expectations can thus be approximated by empirical averages just as for ordinary Monte Carlo. The correlation in the Markov chain however implies that the size n of the Markov chain sample needs to be greater than when independent simulations are used in ordinary Monte Carlo, in order to obtain a given level of accuracy.

4 The Metropolis-Hastings algorithm

Let π denote a complex, multivariate target distribution for a stochastic vector $Z = (Z_1, \ldots, Z_m)$, $m \geq 1$, and let f be the density of π. We shall now consider how one constructs a Markov chain $X = (X_l)_{l \geq 0}$ for which π is the invariant distribution. The constructed chain will then produce samples of π provided that the chain is irreducible and aperiodic. Note that the statespace of X is multidimensional, so that a state $X_l = (X_1^l, \ldots, X_m^l)$, $l \geq 0$ of the Markov chain has components $X_1^l, \ldots, X_m^l$. The initial state X_0 can be chosen rather arbitrarily, but it is advantageous if it belongs to the center of the target distribution π since convergence to the stationary distribution is then faster.

4.1 Metropolis-Hastings algorithm with simultaneous updating of all components

The Metropolis-Hastings algorithm (Hastings, 1971) is given in terms of a proposal kernel q. That is, for any $x \in E$, $q(x, \cdot)$ is a probability density on E. The Metropolis-Hastings algorithm now iterates the following steps:

1. Let $X_l = x = (x_1, \ldots, x_m)$ be the current state of the chain and generate a proposal Y from the proposal density $q(x, \cdot)$.

2. Generate a uniform number U on $[0, 1]$.

3. If
$$U < a(x, Y) = \min\left\{1, \frac{f(Y)q(Y, x)}{f(x)q(x, Y)}\right\}$$
then $X_{l+1} = Y$. Otherwise $X_{l+1} = X_l$.

Note that all components in the current state X_l are updated if the proposal Y is accepted. Note also that if the density $f(Y)$ is small compared to the density $f(x)$ of the current state, then Y will tend to be rejected, so that the chain stays in the stationary distribution.

This is perhaps clearer if one considers the Metropolis algorithm (Metropolis et al., 1953) which is the special case where q is symmetric (i.e. $q(x, z) = q(z, x)$) so that $a(x, y) = \min\{1, f(y)/f(x)\}$

Example 2 cntd. Remember that the AR(1) chain was not convergent when $|\beta| > 1$. We will now "Metropolize" the chain corresponding to the bottom plot in Figure 1. Given the current state $X_l = x$, the next state for the AR(1)-chain was generated from $N(\beta x, \sigma^2)$ with density

$$q(x, z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2\sigma^2}(z - \beta x)^2).$$

Let $\pi = N(-5, 1)$ with density $f(z) = \exp(-(z - (-5))^2/2)/\sqrt{2\pi}$. Instead of just accepting $X_{l+1} = Y$ where Y is generated from $N(\beta x, \sigma^2)$ we now accept or reject Y according to the acceptance probability $a(x, Y)$ (this is of course not exactly a natural way to simulate $N(-5, 1)$). A simulation of the "Metropolized" chain is given in Figure 2. □

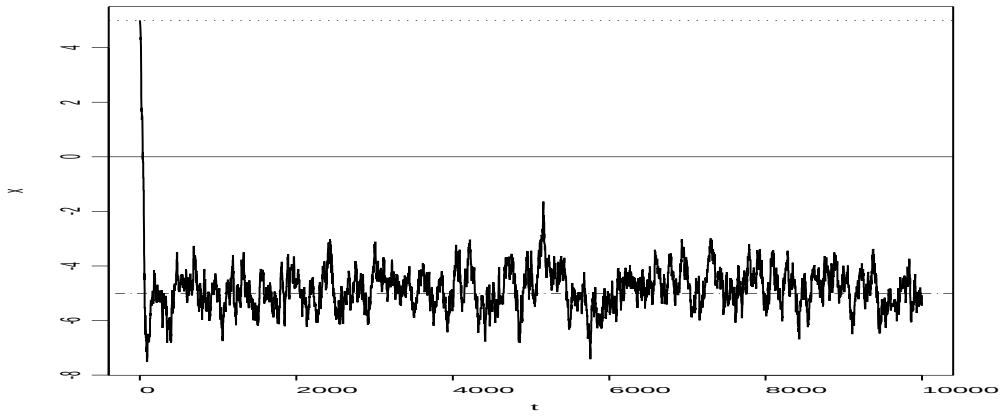


Figure 2: Simulation of "Metropolized" AR(1) chain with stationary distribution given by $N(-5, 1)$.

4.2 Single-site updating Metropolis-Hastings

We now assume that $m > 1$. For a single-site updating Metropolis-Hastings algorithm, the components $X_1^l, \ldots, X_m^l$ are updated individually in a random or systematic order. Often a systematic scheme is chosen so that the components are updated in turn starting with X_1^l, then X_2^l, and so forth.

Assume that the current state is $X_l = (X_1^l, \ldots, X_m^l) = x$ and that the jth component is to be updated. Then the next state X_{l+1} of the chain only differs from X_l on the jth component, i.e. $X_i^{l+1} = X_i^l$, $i \neq j$, and X_{l+1} is generated as follows:

1. A proposal Y_j is generated from a proposal density $q_j(x, \cdot)$. Let $Y = (X_1^l, \ldots, X_{j-1}^l, Y_j, X_{j+1}^l, \ldots, X_m^l)$
2. A uniform variable U on $[0, 1]$ is generated.
3. If
$$U < a_j(x, Y) = \min\left\{1, \frac{f(Y)q_j(Y, x_j)}{f(x)q_j(x, Y_j)}\right\}$$
then $X_{l+1} = Y$. Otherwise $X_{l+1} = X_l$.

Example (Gibbs sampler) The Gibbs sampler is an important example of the single-site Metropolis-Hastings sampler. In this case, $q_j(x, \cdot)$ is simply

the conditional density of Z_j given Z_i, $i \neq j$, i.e.

$$q_j(x, y_j) = f_j(y_j | x_i, i \neq j) = \frac{f(x_1, \ldots, x_{j-1}, y_j, x_{j+1}, \ldots, x_m)}{f_{-j}(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n)},$$

where $f_{-j}(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n)$ is the marginal density of Z_j, $j \neq i$. It is easy to realize that in this case, $a_j(x, y_j) = 1$ so that the proposals are always accepted. □

4.3 Choice of proposal kernel

The target distribution π may be very complex and impossible to sample directly. The advantage of the Metropolis-Hastings algorithm is that it is only required to sample from the proposal kernel Q and we are free to choose any proposal kernel which is easy to sample from, as long as the resulting chain becomes irreducible and aperiodic. Aperiodicity is usually not a problem. It is e.g. enough that there is a positive probability of rejecting generated proposals.

When irreducibility cause problems, it is usually in cases where f, the density of π, is not positive everywhere. Consider e.g. a distribution on $\{0, 1\}^7$, say, where $\pi(x) = 0$ if $\sum_{i=1}^{8} x_i = 4$. Then a single-site updating algorithm is reducible because it can not move from an x with $\sum_{i=1}^{8} x_i < 4$ to an x' with $\sum_{i=1}^{8} x'_i > 4$.

On $\mathbb{R}^2$ one may consider a π whose density is zero outside the balls $b\big((1,1), 0.5\big)$ and $b\big((2,2), 0.5\big)$. In this case the Gibbs sampler becomes reducible (make a drawing) .

5 Some practical issues

The theory of MCMC tells us that the Markov chain eventually will produce samples from the target distribution if we run the chain for sufficiently long time. The difficult question is how long is enough. A hot research topic is "perfect simulation" which in fact answers this question in some cases, but perfect simulation is still a technique for specialists.

A useful way to assess whether convergence is achieved is to consider timeseries of various statistics derived from the Markov chain. Figure 2 e.g. shows a timeseries given by the "Metropolized" AR(1) itself. It seems that convergence has been reached after a burn-in of approximately 100 iterations.

In the Monte Carlo calculations one may feel inclined to discard the first steps of the Markov chain where the chain still have not reached the stationary distribution.

Ordinary timeseries methods are useful for analyzing output from Markov chains. Figure 3 shows autocorrelations estimated from the simulated chain in Figure 2. The chain is in fact highly autocorrelated and the estimate of

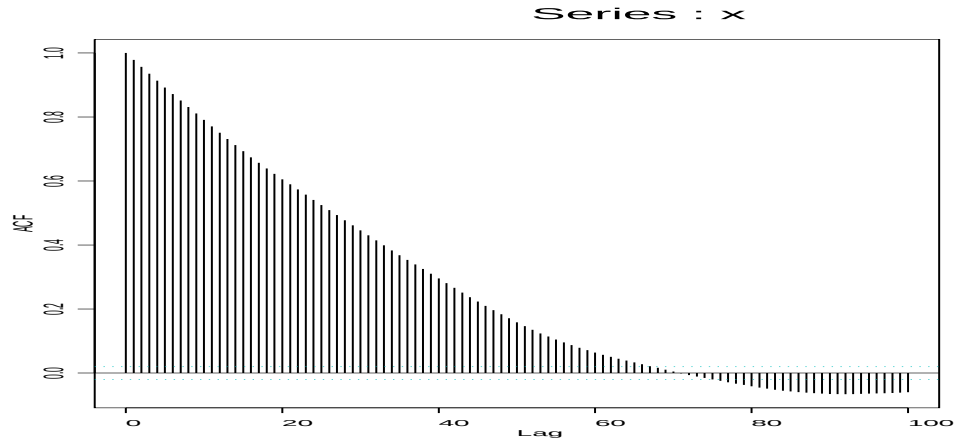


Figure 3: Autocorrelations calculated from the simulation in Figure 2.

the mean (-5) of the stationary distribution accordingly converges slowly, see Figure 4. Compare also with the convergence of the estimate based on independent simulations in Figure 5.

If the MCMC sample is highly autocorrelated, and a way to improve the sampler can not be found, then one may may sometimes wish to subsample the chain, i.e. to create a less correlated sample by retaining only every 10th, say, observation in the original sample. Subsampling throws information away and should in general not be used, but other factors may sometimes render subsampling advantageous. This may e.g. be the case if storage space in the computer is a problem or if the expense of calculating $g(X_l)$ is high, where $g(\cdot)$ is the function whose mean $E(g(Z))$ is to be estimated.

References

Chan, K. S. & Geyer, C. J. (1994). Discussion of the paper 'Markov chains for exploring posterior distributions' by Luke Tierney. *Annals of Statistics*

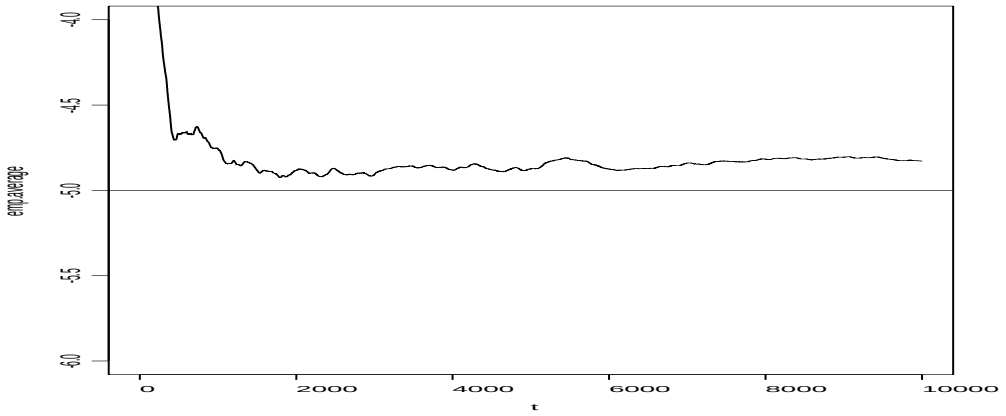


Figure 4: Convergence of empirical average as a function of sample size ("Metropolized" AR(1)).

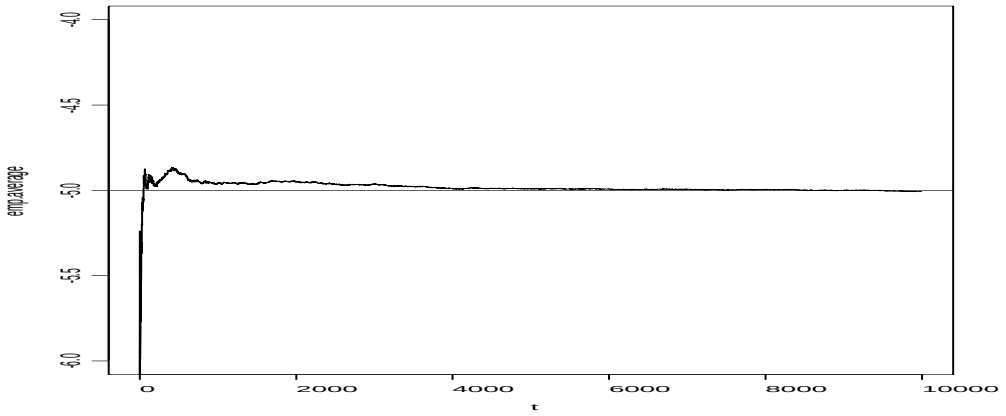


Figure 5: Convergence of empirical average as a function of sample size (independent simulations).

22, 1747–1747.

Gamerman, D. (1997). *Markov Chain Monte Carlo*. Chapman and Hall, London.

Gilks, W. R., Richardson, S. & Spiegelhalter, D. J. (1996). *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London.

Hastings, W. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **57**, 97–109.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics* **21**, 1087–1092.

Robert, C. P. & Casella, G. (1999). *Monte Carlo Statistical Methods*. Springer-Verlag, New York.

Tierney, L. (1994). Markov chains for exploring posterior distributions. *Annals of Statistics* **22**, 1701–1728.

Exercises for the DINA Research School Workshop on Monte Carlo methods for hierarchical models

Rasmus Waagepetersen
Institute of Mathematical Sciences
Aalborg University

October 25, 2003

1 Introduction

The exercises are supposed to be solved using either `R` or `BUGS`. Many of the exercises are one- or two-dimensional "toy examples" in order to keep the programming simple. This creates a bit of a paedagogical conflict in a course on Monte Carlo methods since in one dimension basically any method works while Monte Carlo methods prevail in high-dimensional problems.

2 Some comments on `R`

`R` is a free-ware version of Splus. A wide range of statistical methods are implemented in `R` and `R` is furthermore a very flexible programming language where new applications can easily be developed.

2.1 Entering commands

In your pc version commands can be executed via the command line. Longer sequences of commands or implementations of new functions can be written in a separate file. If the commands are stored in a file, `cmd.R` say, then the command line statement `source(‘‘cmd.R’’)` will execute the commands

in `cmd.R` and load function code contained in `cmd.R`. You can also open `cmd.R` in your favourite text editor and use copy/paste to paste commands or sequences of commands into the command line.

2.2 Introduction and help

You may find it useful to run the commands in `basic.R` to learn more about some basic features of R. Help can be obtained by clicking the help-button. For help on a specific topic, say the function `scan`, you may alternatively just type `help(scan)` on the command line.

3 Exercises

You may either program the solutions from scratch or consider the programs in the `.R` or `.odc` files associated to each exercise. However, to encourage you to work actively with the code you will often need to either first correct bugs in the programs or modify the programs before you get the correct solution.

3.1 Exercises for Lecture I

1 (`exercise1.R`)
Suppose $U \sim N(0,1)$ and $Y|U = u \sim \text{Poisson}(\exp(\beta + U))$ (recall that Poisson(λ) has density $f(y;\lambda) = \exp(-\lambda)\lambda^y/y!$). Assume that $Y = 8$ is observed.

a) In the same plot draw the density $f(u) = \exp(-u^2/2)/\sqrt{2\pi}$ of $N(0,1)$ and $f(8;\exp(\beta+u))$ as a function of u and for various β-values - why can the variance of $f(8;\exp(\beta+U)$ be large for some β ?. Also plot the product $f(8;\exp(\beta+u)f(u)$ of the two densities. Can you tell which values of β yield small likelihoods ?
b) Compute and plot the marginal likelihood of β using numerical quadrature (Use e.g. the `R` function `integrate`).
c) Compute and plot a Laplace-approximation of the likelihood (note that you obtain as a biproduct a normal distribution with mean and variance approximately equal to the conditional mean and variance of U given $Y = y$).
d) Compute and plot a simple Monte Carlo approximation of the likelihood and compute an estimate of the Monte Carlo error. Compare results obtained

with different numbers of simulations.
e) Repeat d) - c) but in the situations where 10 observations 8, 18, 5, 7, 10, 9, 9, 6, 7, 10 are available (i.e. $f(8; \exp(\beta + u))$ is replaced by the product

$$\prod_{i=1}^{10}[f(y_i; \exp(\beta + u))$$

of conditional densities for these observations). Note that we now have to take care of numerical problems since the product of 10 Poisson densities may take very small values. One possibility is to stabilize by dividing with the product of Poisson densities evaluated at $\hat{\lambda} = \bar{y}$ (the MLE). Does the accuracy of the Laplace approximation improve or worsen when more observations are available ?

2 (`exercise2.R`) (Pig growth)

The *R*-package `nlme` can be used to compute maximum likelihood estimates of parameters in non-linear mixed models. This package use a specialized numerical integration method called Gaussian quadrature.

a) Consider a Gompertz model

$$y = \exp(a - (a - \log(v0)) \exp(-k * t)) + \epsilon$$

for pig growth where k is the growth rate, t is time in days, $v0$ corresponds to the initial weight of pig, and ϵ is $N(0, \sigma^2)$ residual noise. Fit a Gompertz model with a random growth coefficent to the growth data (`growth.txt`). Also fit a Gompertz model with a non random growth coefficient. Why is the estimated residual variance larger for the latter model ?
b) Use the fitted models to simulate and plot growth curves for the five first pigs. Does the simulated curves resemble the data ?

3.2 Exercises for Lecture II

3 (`exercise3.R`) (continuation of Exercise 1)

This exercise considers importance sampling approximation of the likelihood

using a t-distribution with mean and variance from the Laplace approximation obtained in Exercise 1 e).

a) In the same plot draw the density of the importance t-distribution and of the joint density

$$\exp(g(u)) = \prod_{i=1}^{10} [f(y_i; \exp(\beta + u))/f(y_i; \bar{y})] f(u)$$

of the observations and the random effect (here we have for numerical stability divided with $f(y_i; \bar{y})$). How does the importance sampling distribution depend on the value of β ?
b) Compute an importance sampling approximation of the likelihood. What happens if you use the importance sampling t-distribution obtained for, say $\beta = 4$ to compute the likelihood for all the other values of β ?
b) Compute the first and second order derivatives of the importance sampling approximation of the log likelihood and maximize the log likelihood using Newton-Raphson.

4 (`exercise4.R`) (continuation of Exercise 1)

Construct a random walk Metropolis sampler which simulates from $U|Y = y$. Compare the Monte Carlo estimate of the conditional distribution with the approximate normal distribution obtained in Exercise 1 c). A rule of thumb states that the optimal acceptance rate is around 25 %. Use different proposal variances (leading to different acceptance rates) and compare the estimated autocorrelations (use `acf` to compute autocorrelations).

5 (`exercise5.R`) (Gibbs sampler for bivariate normal)
Consider a bivariate normal distribution

$$(U_1, U_2) \sim N\big((0,0), \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}\big)$$

Note that the conditional distributions of $U_1|U_2 = u_2$ and $U_2|U_1 = u_1$ are

$$N(\rho u_2, 1 - \rho^2) \text{ and}$$
$$N(\rho u_1, 1 - \rho^2),$$

respectively.

Run a Gibbs sampler for (U_1, U_2) for values of $\rho = 0, 0.5$, and 0.95. Plot the trajectory of the simulated pairs of (U_1, U_2). Compute autocorrelation for the simulated chain using `acf`. Why are the chains more auto-correlated (slowly mixing) when ρ is large ?

3.3 Exercises for Lecture III

6 (`simple_ex.odc`) (continuation of Exercise 1)

As a warm up for `BUGS`, try to simulate from the conditional distribution of $U|Y = y$ using `BUGS`.

7 (`cherries.odc,cherriesdat.odc`) (Bayesian analysis for cherries)

Perform a Bayesian analysis of the cherry data using `BUGS`. Assess convergence by considering timeseries and compute posterior means and variances of treatment effects and variance components.

3.4 Exercises for Lecture IV

8 (`exercise8.R`) (sensitivity analysis)

Use importance sampling to asses the sensitivity of the posterior results from Exercise 7 to the choice of prior.

9 (`cherriesbinary.odc`) (non-Gaussian random effects, continuation of Exercise 7)

Try to replace the Gaussian random branch effects with binary random effects. Try also to replace an observation with `NA` (so that it is missing) and then predict that observation with `BUGS`.

A R - Programs for exercises *Rasmus Waagepetersen*

A.1 Initial

A.1.1 basic.R

```
#In R the assignment operator is <-. Assignments within

#functions are local. A global assignment within a function may be

#performed using <<- (this is a bit dangerous programming style).


#create a vector x consisting of the numbers 1 2 and 3

x<-c(1,2,3)

x #x is printed...

#add 3 to all entries in x

z<-x+3

z

#multiply last entry in x by 7

x[3]<-x[3]*7

x

#create new vector by combining z and x

zx<-c(z,x)

zx


#create a 3 by 2 matrix with z in the first column and x in the second

A<-matrix(zx,3,2)

#take a look at A

A

#take a look at first row in A

A[1,]

#take a look at second column in A

A[,2]
```

```
#take a look at last 2 rows in A

A[c(2,3),]

#equivalent:

A[2:3,]


#create a list containing x, z, and A

l<-list(x=x,A=A,z=z)

#extract x from l

l$x


#compute square root of all entries in x

s<-sqrt(x)

s


#construct vector of 4 10's

ff<-rep(10,4)

ff
```

A.2 Exercises for Lecture I

A.2.1 Exercise1.R

```
#Exercise 1

beta=1
y=c(8,18,5,7,10,9,9,6,7,10)

#a)
par(mfrow=c(2,2))
u=c(-100:100)/20
beta=-2
lambda=exp(u+beta)
plot(u,dnorm(u),type="l",lty=1)
lines(u,dpois(8,lambda),lty=2)
lines(u,dpois(8,lambda)*dnorm(u),lty=3)

beta=0
```

```
lambda=exp(u+beta)
plot(u,dnorm(u),type="l",lty=1)
lines(u,dpois(8,lambda),lty=2)
lines(u,dpois(8,lambda)*dnorm(u),lty=3)

beta=2
lambda=exp(u+beta)
plot(u,dnorm(u),type="l",lty=1)
lines(u,dpois(8,lambda),lty=2)
lines(u,dpois(8,lambda)*dnorm(u),lty=3)

#b
par(mfrow=c(1,1))
integrand=function(u){
  return(dpois(8,exp(beta+u))*dnorm(u))
}
betas=c(-10:40)/10
quadlike=rep(0,length(betas))
for (i in 1:length(betas)){
  beta=betas[i]
  quadlike[i]=integrate(integrand,-10,10)$value
}
plot(betas,quadlike,type="l")

#c
#Oups, there are two bugs in the procedures minusg() and Laplaceapprox()
#the function minusg() implements -g(u).
#The minimum of -g(u) which is the maximum of g(u) is used in the Laplace approximation.
minusg=function(u){
  return(-8*(beta+u)+exp(beta+u)+sum(log(c(1:8)))+log(dnorm(u)))
}
Laplaceapprox=function(y,uhat,beta){
  return(exp(-minusg(uhat))*sqrt(2*pi/(exp(beta+uhat))))
}
laplacelike=rep(0,length(betas))
uhat=rep(0,length(betas))
for (i in 1:length(betas)){
  beta=betas[i]
  uhat[i]=nlm(minusg,2-beta)$estimate
  laplacelike[i]=Laplaceapprox(8,uhat[i],betas[i])
}
plot(betas,quadlike,type="l",lty=1)
lines(betas,laplacelike,lty=2)
#similar results obtained

#d
#an error has occurred in LSMC below
LSMC=function(y,beta,M){
  u=rpois(M)
  lambda=exp(beta+u)
  return(list(mcestimate=mean(dpois(y,lambda)),mcstderr=sqrt(var(dpois(y,lambda)/M))))
}

SMClike=rep(0,length(betas))
SMCerr=rep(0,length(betas))
M=1000
```

```
for (i in 1:length(betas)){
  temp=LSMC(8,betas[i],M)
  SMClike[i]=temp$mcestimate
  SMCerr[i]=temp$mcstderr
}

plot(betas,quadlike,type="l",lty=1)
lines(betas,SMClike,lty=2)

#e repeat a-d with more observations

#a)
u=c(-100:100)/20

#you need to correct the proceduce densityprod which gives the product of standardised poisson den
densityprod=function(lambda){
  #we standardise with poisson density at lambda=mean(y) for numerical stability
  return(exp( sum(y)*((lambda)-log(mean(y))) - length(y)*(lambda - mean(y))))
}

par(mfrow=c(2,2))

beta=-2
lambda=exp(u+beta)
temp=rep(0,length(lambda))
for (i in 1:length(lambda))
  temp[i]=densityprod(lambda[i])
plot(u,temp,type="l",lty=1)
lines(u,dnorm(u),lty=2)
lines(u,temp*dnorm(u),lty=3)

beta=0
lambda=exp(u+beta)
temp=rep(0,length(lambda))
for (i in 1:length(lambda))
  temp[i]=densityprod(lambda[i])
plot(u,temp,type="l",lty=1)
lines(u,dnorm(u),lty=2)
lines(u,temp*dnorm(u),lty=3)

beta=2
lambda=exp(u+beta)
temp=rep(0,length(lambda))
for (i in 1:length(lambda))
  temp[i]=densityprod(lambda[i])
plot(u,temp,type="l",lty=1)
lines(u,dnorm(u),lty=2)
lines(u,temp*dnorm(u),lty=3)

#b
par(mfrow=c(1,1))

integrand=function(u){
  return(densityprod(exp(u+beta))*dnorm(u))
}
betas=c(-10:40)/10
```

```
quadlike=rep(0,length(betas))
low=-10
up=10
for (i in 1:length(betas)){
  beta=betas[i]
  quadlike[i]=integrate(integrand,low,up)$value
}
plot(betas,quadlike,type="l")


#c
#there are still errors in minusg
minusg=function(u){
  #again we standardise with poisson density at lambda=mean(y) for numerical stability
  return(-sum(y)*((beta)-log(mean(y)))+length(y)*(exp(beta+u)-mean(y))-log(dnorm(u)))
}

plotminusg=function(){
  u=c(-50:50)/10
  temp=length(u)
  for (i in 1:length(u))
    temp[i]=minusg(u[i])
  plot(u,temp)
}

Laplaceapprox=function(y,uhat,beta){
  return(exp(-minusg(uhat)+0.5*log(2*pi)-0.5*log(length(y)*exp(beta+uhat)+1)))
}
laplacelike=rep(0,length(betas))
uhat=rep(0,length(betas))
for (i in 1:length(betas)){
  beta=betas[i]
  uhat[i]=nlm(minusg,2-beta)$estimate
  laplacelike[i]=Laplaceapprox(y,uhat[i],betas[i])
}
plot(betas,quadlike,type="l",lty=1)
lines(betas,laplacelike,lty=2)
plot(quadlike,laplacelike)
abline(c(0,1))

#d
LSMC=function(y,beta,M){
  u=rnorm(M)
  lambda=exp(beta+u)
  return(list(mcestimate=mean(densityprod(lambda)),mcstderr=sqrt(var(densityprod(lambda)/M))))
}

SMClike=rep(0,length(betas))
SMCerr=rep(0,length(betas))
M=1000
for (i in 1:length(betas)){
  temp=LSMC(8,betas[i],M)
  SMClike[i]=temp$mcestimate
  SMCerr[i]=temp$mcstderr
}
```

```
plot(betas,quadlike,type="l",lty=1)
lines(betas,SMClike,lty=2)
```

A.2.2 Exercise1_correct.R

```
#Exercise 1

beta=1
y=c(8,18,5,7,10,9,9,6,7,10)

#a)
par(mfrow=c(2,2))
u=c(-100:100)/20
beta=-2
lambda=exp(u+beta)
plot(u,dnorm(u),type="l",lty=1)
lines(u,dpois(8,lambda),lty=2)
lines(u,dpois(8,lambda)*dnorm(u),lty=3)

beta=0
lambda=exp(u+beta)
plot(u,dnorm(u),type="l",lty=1)
lines(u,dpois(8,lambda),lty=2)
lines(u,dpois(8,lambda)*dnorm(u),lty=3)

beta=2
lambda=exp(u+beta)
plot(u,dnorm(u),type="l",lty=1)
lines(u,dpois(8,lambda),lty=2)
lines(u,dpois(8,lambda)*dnorm(u),lty=3)

#b
par(mfrow=c(1,1))
integrand=function(u){
  return(dpois(8,exp(beta+u))*dnorm(u))
}
betas=c(-10:40)/10
quadlike=rep(0,length(betas))
for (i in 1:length(betas)){
  beta=betas[i]
  quadlike[i]=integrate(integrand,-10,10)$value
}
plot(betas,quadlike,type="l")

#c
#Oups, there are two bugs in the procedures minusg() and Laplaceapprox()
#the function minusg() implements -g(u).
#The minimum of -g(u) which is the maximum of g(u) is used in the Laplace approximation.
minusg=function(u){
  return(-8*(beta+u)+exp(beta+u)+sum(log(c(1:8)))-log(dnorm(u)))
}
Laplaceapprox=function(y,uhat,beta){
  return(exp(-minusg(uhat))*sqrt(2*pi/(exp(beta+uhat)+1)))
}
```

```
laplacelike=rep(0,length(betas))
uhat=rep(0,length(betas))
for (i in 1:length(betas)){
  beta=betas[i]
  uhat[i]=nlm(minusg,2-beta)$estimate
  laplacelike[i]=Laplaceapprox(8,uhat[i],betas[i])
}
plot(betas,quadlike,type="l",lty=1)
lines(betas,laplacelike,lty=2)
#similar results obtained

#d
#an error has occurred in LSMC below
LSMC=function(y,beta,M){
  u=rnorm(M)
  lambda=exp(beta+u)
  return(list(mcestimate=mean(dpois(y,lambda)),mcstderr=sqrt(var(dpois(y,lambda)/M))))
}

SMClike=rep(0,length(betas))
SMCerr=rep(0,length(betas))
M=1000
for (i in 1:length(betas)){
  temp=LSMC(8,betas[i],M)
  SMClike[i]=temp$mcestimate
  SMCerr[i]=temp$mcstderr
}

plot(betas,quadlike,type="l",lty=1)
lines(betas,SMClike,lty=2)

#e repeat a-d with more observations

#a)
u=c(-100:100)/20

#you need to correct the proceduce densityprod which gives the product of standardised poisson den
densityprod=function(lambda){
  #we standardise with poisson density at lambda=mean(y) for numerical stability
  return(exp( sum(y)*(log(lambda)-log(mean(y))) - length(y)*(lambda - mean(y))))
}

par(mfrow=c(2,2))

beta=-2
lambda=exp(u+beta)
temp=rep(0,length(lambda))
for (i in 1:length(lambda))
  temp[i]=densityprod(lambda[i])
plot(u,temp,type="l",lty=1)
lines(u,dnorm(u),lty=2)
lines(u,temp*dnorm(u),lty=3)

beta=0
lambda=exp(u+beta)
temp=rep(0,length(lambda))
```

```
for (i in 1:length(lambda))
  temp[i]=densityprod(lambda[i])
plot(u,temp,type="l",lty=1)
lines(u,dnorm(u),lty=2)
lines(u,temp*dnorm(u),lty=3)

beta=2
lambda=exp(u+beta)
temp=rep(0,length(lambda))
for (i in 1:length(lambda))
  temp[i]=densityprod(lambda[i])
plot(u,temp,type="l",lty=1)
lines(u,dnorm(u),lty=2)
lines(u,temp*dnorm(u),lty=3)

#b
par(mfrow=c(1,1))

integrand=function(u){
  return(densityprod(exp(u+beta))*dnorm(u))
}
betas=c(-10:40)/10
quadlike=rep(0,length(betas))
low=-10
up=10
for (i in 1:length(betas)){
  beta=betas[i]
  quadlike[i]=integrate(integrand,low,up)$value
}
plot(betas,quadlike,type="l")


#c
#there are still errors in minusg
minusg=function(u){
  #again we standardise with poisson density at lambda=mean(y) for numerical stability
  return(-sum(y)*((beta+u)-log(mean(y)))+length(y)*(exp(beta+u)-mean(y))-log(dnorm(u)))
}

plotminusg=function(){
  u=c(-50:50)/10
  temp=length(u)
  for (i in 1:length(u))
    temp[i]=minusg(u[i])
  plot(u,temp)
}

Laplaceapprox=function(y,uhat,beta){
  return(exp(-minusg(uhat)+0.5*log(2*pi)-0.5*log(length(y)*exp(beta+uhat)+1)))
}
laplacelike=rep(0,length(betas))
uhat=rep(0,length(betas))
for (i in 1:length(betas)){
  beta=betas[i]
  uhat[i]=nlm(minusg,2-beta)$estimate
  laplacelike[i]=Laplaceapprox(y,uhat[i],betas[i])
```

```
}
plot(betas,quadlike,type="l",lty=1)
lines(betas,laplacelike,lty=2)
plot(quadlike,laplacelike)
abline(c(0,1))

#d
LSMC=function(y,beta,M){
  u=rnorm(M)
  lambda=exp(beta+u)
  return(list(mcestimate=mean(densityprod(lambda)),mcstderr=sqrt(var(densityprod(lambda)/M))))
}

SMClike=rep(0,length(betas))
SMCerr=rep(0,length(betas))
M=1000
for (i in 1:length(betas)){
  temp=LSMC(8,betas[i],M)
  SMClike[i]=temp$mcestimate
  SMCerr[i]=temp$mcstderr
}

plot(betas,quadlike,type="l",lty=1)
lines(betas,SMClike,lty=2)
```

A.2.3 Exercise2.R

```
#read data
growth=read.table("growth.txt",header=T)
#make variables in growth known
attach(growth)
library(nlme)

#use first observed weight v0 as offset for each pig
v0=vgt
for (i in 1:length(v0))
  v0[i]=vgt[vejenr==1 & gris==gris[i]]

#fit nonlinear model without random effects. k is growth rate.
fit=nls(vgt~exp(a-(a-log(v0))*exp(-k*dage)),start=list(a=log(100),k=0.1))
summary(fit)
#estimated a and k are 4.836192 and 0.029655. Estimated residual standard error is 3.524.


#fit nonlinear mixed model with random growth rate k
fitmixed=nlme(vgt~exp(a-(a-log(v0))*exp(-k*dage)),
              fixed=a+k~1,random=k~1,groups=~gris,start=c(4.83,0.01))
summary(fitmixed)
#estimated mean and std. dev for k is 0.019509 and 0.002636722
#estimated a is 5.094369 estimated residual std. dev. is 1.978583

#compute simulated curves
par(mfrow=c(1,3))
```

```
#first non-linear model without random effects

a=4.836192
k=0.029655
curve=exp(a-(a-log(v0))*exp(-k*dage))+rnorm(length(dage),sd=3.514)

#plot simulated curves based on nonlinear model for five first pigs
grisno=unique(gris)
plot(0,0,xlim=c(min(dage[1:32]),max(dage[1:32])),
         ylim=c(min(curve),max(curve)),type="n",xlab="",ylab="")
for (i in 1:5)
  lines(dage[gris==grisno[i]],curve[gris==grisno[i]],lty=i)

#secondly mixed non-linear models
a=5.094369

#simulate k for each pig
k=rnorm(length(grisno),0.019509,0.002636722)

#the value of k for a pig is repeated for each of the pigs observations
ks=rep(0,length(v0))
for (i in 1:length(grisno))
  ks[gris==grisno[i]]=k[i]

curve=exp(a-(a-log(v0))*exp(-ks*dage))+rnorm(length(dage),sd=1.978583)

#plot simulated curves based on mixed model for five first pigs
plot(0,0,xlim=c(min(dage[1:32]),max(dage[1:32])),
         ylim=c(min(curve),max(curve)),type="n",xlab="",ylab="")
for (i in 1:5)
  lines(dage[gris==grisno[i]],curve[gris==grisno[i]],lty=i)

#finally plot observed curves for five first pigs

plot(0,0,xlim=c(min(dage[1:32]),max(dage[1:32])),
         ylim=c(min(curve),max(curve)),type="n",xlab="",ylab="")
for (i in 1:5)
  lines(dage[gris==grisno[i]],vgt[gris==grisno[i]],lty=i)
```

A.3 Exercises for Lecture II

A.3.1 Exercise3.R

```
#Exercise 2
y=c(8,18,5,7,10,9,9,6,7,10)

#a

impsampquotient=function(v){
  return(exp( -minusg(v)-log( dt((v-mu)/sigma,nu)/sigma)) )
}

#plot numerator and denominator in importance sampling quotiont
nu=1000
beta=-2
```

```
mu=nlm(minusg,2-beta)$estimate
sigma=1/sqrt(length(y)*exp(beta+mu)+1)
plot(u,exp(-minusg(u)),type="l",lty=1)
lines(u,dt((u-mu)/sigma,nu)/sigma,lty=2)

V=rt(1000,nu)
impsamplelike=rep(0,length(betas))
for (i in 1:length(betas)){
  beta=betas[i]
  mu=nlm(minusg,2-beta)$estimate
  sigma=1/sqrt(length(y)*exp(beta+mu)+1)
  v=mu+sigma*V
  impsamplelike[i]=mean(impsampquotient(v))
}
#plot importance sampling estimate of likelihood with the likelihood computed using numerical quad
plot(betas,log(quadlike),type="l",lty=1)
lines(betas,log(impsamplelike),lty=2)

#now use one importance sampling distribution for all betas
impsamplelike=rep(0,length(betas))
beta=4
mu=nlm(minusg,2-beta)$estimate
sigma=1/sqrt(length(y)*exp(beta+mu)+1)
v=mu+sigma*V
for (i in 1:length(betas)){
  beta=betas[i]
  impsamplelike[i]=mean(impsampquotient(v))
}
plot(betas,log(quadlike),type="l",lty=1)
lines(betas,log(impsamplelike),lty=2)
#that does not work well.....


#compute derivative  and second derivative of wrt beta of importance sampling approximation of log

derivatives=function(v){
  Lbeta=mean(impsampquotient(v))

  dLbeta=mean(impsampquotient(v)*(sum(y)-length(y)*exp(beta+v)))
  u=dLbeta/Lbeta

  d2Lbeta=mean(impsampquotient(v)*((sum(y)-length(y)*exp(beta+v))^2-length(y)*exp(beta+v)))


  du=d2Lbeta/Lbeta-u^2
  i=-du

  return(list(u=u,i=i))
}



#newton-raphson iteration large number of samples required to get sufficiently precise Monte Carlo
betanext=0
V=rt(100000,nu)
```

```
beta=betanext
mu=nlm(minusg,2-beta)$estimate
sigma=1/sqrt(length(y)*exp(beta+mu)+1)
v=mu+sigma*V
deriv=derivatives(v)
print(c(deriv$u,deriv$i))
#next beta
betanext=beta+deriv$u/deriv$i
print(betanext)
```

A.3.2 Exercise4.R

```
#Exercise 4

#the implementation metrop of the metropolis sampler suffers from a bug.
metrop=function(uinit,beta,samplelength,propvar){
  usample=rep(0,samplelength)
  usample[1]=uinit

  acceptrate=0

  for (i in 2:samplelength){
    #generate proposal
    uprop=rnorm(1,usample[i-1],sqrt(propvar))
    #compute log  metropolis ratio
    logmh=sum(y)*(usample[i-1]-uprop)-length(y)*(exp(uprop+beta)-exp(usample[i-1]+beta))+log(dnorm
    #notice: for numerical reason it is better to compute on the log scale
    if (log(runif(1))<logmh){
      acceptrate=acceptrate+1
      usample[i]=uprop
    } else
      usample[i]=usample[i-1]
  }
  print(c("accept rate:",acceptrate/samplelength))
  return(usample)
}

#generate sample
beta=0
sample=metrop(4,beta,10000,2)
par(mfrow=c(1,2))
#plot timeseries
#postscript(file="quick.ps",horizontal=F)
#postscript(file="slow.ps",horizontal=F)
plot(sample,type="l")
#dev.off()
#plot autocorrelation
library(ts)
#postscript(file="quick_acf.ps",horizontal=F)
#postscript(file="slow_acf.ps",horizontal=F)
acf(sample)
#dev.off()
#MCMC estimate of conditional mean and variance
mean(sample[10:10000])
var(sample[10:10000])
```

```
#laplace approximation of conditional mean and variance
mu=nlm(minusg,2-beta)$estimate
sigma2=1/(length(y)*exp(beta+mu)+1)
mu
sigma2

# derivatives when conditional samples are used
derivatives.mcmc=function(v){

  u=mean((sum(y)-length(y)*exp(beta+v)))

  i=-mean(-length(y)*exp(beta+v))-var(sum(y)-length(y)*exp(beta+v))

  i2=-1*(-u^2+mean(-length(y)*exp(beta+v))+mean((sum(y)-length(y)*exp(beta+v))^2))
  print(i2)
  return(list(u=u,i=i))
}

#newton-raphson iteration
betanext=2

beta=betanext
sample=metrop(0,beta,50000,0.2)
deriv=derivatives.simple(sample[10:50000])
print(c(deriv$u,deriv$i))
#next beta
betanext=beta+deriv$u/deriv$i
print(betanext)
#due to Monte Carlo error the procedure ends up oscillating around the mle
#instead of generating a new sample in each iteration it is more efficient
#to reuse the sample from the previous iteration using importance sampling
#(as long as the current beta value is not too far from the beta value used
#for generating the sample).
```

A.3.3 Exercise4_correct.R

```
#Exercise 4

#the implementation metrop of the metropolis sampler suffers from a bug.
metrop=function(uinit,beta,samplelength,propvar){
  usample=rep(0,samplelength)
  usample[1]=uinit

  acceptrate=0

  for (i in 2:samplelength){
    #generate proposal
    uprop=rnorm(1,usample[i-1],sqrt(propvar))
    #compute log  metropolis ratio
    logmh=sum(y)*(uprop-usample[i-1])-length(y)*(exp(uprop+beta)
          -exp(usample[i-1]+beta))+log(dnorm(uprop))-log(dnorm(usample[i-1]))
    #notice: for numerical reason it is better to compute on the log scale
    if (log(runif(1))<logmh){
      acceptrate=acceptrate+1
      usample[i]=uprop
```

```
    } else
      usample[i]=usample[i-1]
  }
  print(c("accept rate:",acceptrate/samplelength))
  return(usample)
}

#generate sample
beta=0
sample=metrop(4,beta,10000,2)
par(mfrow=c(1,2))
#plot timeseries
#postscript(file="quick.ps",horizontal=F)
#postscript(file="slow.ps",horizontal=F)
plot(sample,type="l")
#dev.off()
#plot autocorrelation
library(ts)
#postscript(file="quick_acf.ps",horizontal=F)
#postscript(file="slow_acf.ps",horizontal=F)
acf(sample)
#dev.off()
#MCMC estimate of conditional mean and variance
mean(sample[10:10000])
var(sample[10:10000])
#laplace approximation of conditional mean and variance
mu=nlm(minusg,2-beta)$estimate
sigma2=1/(length(y)*exp(beta+mu)+1)
mu
sigma2

# derivatives when conditional samples are used
derivatives.mcmc=function(v){

  u=mean((sum(y)-length(y)*exp(beta+v)))

  i=-mean(-length(y)*exp(beta+v))-var(sum(y)-length(y)*exp(beta+v))

  i2=-1*(-u^2+mean(-length(y)*exp(beta+v))+mean((sum(y)-length(y)*exp(beta+v))^2))
  print(i2)
  return(list(u=u,i=i))
}

#newton-raphson iteration
betanext=2

beta=betanext
sample=metrop(0,beta,50000,0.2)
deriv=derivatives.simple(sample[10:50000])
print(c(deriv$u,deriv$i))
#next beta
betanext=beta+deriv$u/deriv$i
print(betanext)
#due to Monte Carlo error the procedure ends up oscillating around the mle
#instead of generating a new sample in each iteration it is more efficient
#to reuse the sample from the previous iteration using importance sampling
```

```
#(as long as the current beta value is not too far from the beta value used
#for generating the sample).
```

A.3.4 Exercise5.R

```
#Exercise 2
y=c(8,18,5,7,10,9,9,6,7,10)

#a

impsampquotient=function(v){
  return(exp( -minusg(v)-log( dt((v-mu)/sigma,nu)/sigma)) )
}

#plot numerator and denominator in importance sampling quotiont
nu=1000
beta=-2
mu=nlm(minusg,2-beta)$estimate
sigma=1/sqrt(length(y)*exp(beta+mu)+1)
plot(u,exp(-minusg(u)),type="l",lty=1)
lines(u,dt((u-mu)/sigma,nu)/sigma,lty=2)

V=rt(1000,nu)
impsamplelike=rep(0,length(betas))
for (i in 1:length(betas)){
  beta=betas[i]
  mu=nlm(minusg,2-beta)$estimate
  sigma=1/sqrt(length(y)*exp(beta+mu)+1)
  v=mu+sigma*V
  impsamplelike[i]=mean(impsampquotient(v))
}
#plot importance sampling estimate of likelihood with the likelihood computed using numerical quad
plot(betas,log(quadlike),type="l",lty=1)
lines(betas,log(impsamplelike),lty=2)

#now use one importance sampling distribution for all betas
impsamplelike=rep(0,length(betas))
beta=4
mu=nlm(minusg,2-beta)$estimate
sigma=1/sqrt(length(y)*exp(beta+mu)+1)
v=mu+sigma*V
for (i in 1:length(betas)){
  beta=betas[i]
  impsamplelike[i]=mean(impsampquotient(v))
}
plot(betas,log(quadlike),type="l",lty=1)
lines(betas,log(impsamplelike),lty=2)
#that does not work well.....


#compute derivative  and second derivative of wrt beta of importance sampling approximation of log

derivatives=function(v){
  Lbeta=mean(impsampquotient(v))
```

```
  dLbeta=mean(impsampquotient(v)*(sum(y)-length(y)*exp(beta+v)))
  u=dLbeta/Lbeta

  d2Lbeta=mean(impsampquotient(v)*((sum(y)-length(y)*exp(beta+v))^2-length(y)*exp(beta+v)))


  du=d2Lbeta/Lbeta-u^2
  i=-du

  return(list(u=u,i=i))
}



#newton-raphson iteration large number of samples required to get sufficiently precise Monte Carlo
betanext=0
V=rt(100000,nu)

beta=betanext
mu=nlm(minusg,2-beta)$estimate
sigma=1/sqrt(length(y)*exp(beta+mu)+1)
v=mu+sigma*V
deriv=derivatives(v)
print(c(deriv$u,deriv$i))
#next beta
betanext=beta+deriv$u/deriv$i
print(betanext)
```

A.4 Exercises for Lecture III

A.4.1 Simple_ex.odc

```
model{
        u~dnorm(0.0,1)
        beta<-0
        for (i in 1:10){
                log(ld[i])<-beta+u
                y[i]~dpois(ld[i])
                }
}

list(y=c(8,18,5,7,10,9,9,6,7,10))

list(u=0)
```

A.4.2 Cherries.odc

```
model
    {
     tau2parcelrec~dgamma(0.001,0.001)
     tau2branchrec~dgamma(0.001,0.001)
     for (i in 1:20){
         parcel[i]~dnorm(0.0,tau2parcelrec)
         }
```

```
     for (i in 1:60){
         branch[i]~dnorm(0.0,tau2branchrec)
         }
     for( i in 1 : 953 ) {
         dummy[i]<-no[i]
         y[i] ~ dbin(p[i],1)
         logit(p[i]) <- beta1 * g1[i]+ beta2*g2[i]+beta3*g3[i]
                       +beta4*g4[i]+beta5*g5[i]
                       +branch[branchno[i]]+parcel[parcelno[i]]
         }
     beta1 ~ dnorm(0.0,1.0E-6)
     beta2~ dnorm(0.0,1.0E-6)
     beta3 ~ dnorm(0.0,1.0E-6)
     beta4 ~ dnorm(0.0,1.0E-6)
     beta5~ dnorm(0.0,1.0E-6)
     tau2parcel<-1/tau2parcelrec
     tau2branch<-1/tau2branchrec
     }

Initial values

list(branch=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0),
     parcel=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
     beta1=0,beta2=0,beta3=0,beta4=0,beta5=0,
     tau2branchrec=1,tau2parcelrec=1)


Model where branch also appears as an systematic factor with
three levels


model
   {
   tau2parcelrec~dgamma(0.001,0.001)
   tau2branchrec~dgamma(0.001,0.001)
   for (i in 1:20){
       parcel[i]~dnorm(0.0,tau2parcelrec)
       }
   for (i in 1:60){
       branch[i]~dnorm(0.0,tau2branchrec)
       }
   for( i in 1 : 953 ) {
      dummy[i]<-no[i]
      y[i] ~ dbin(p[i],1)
      logit(p[i]) <- beta1 * g1[i]+ beta2*g2[i]+beta3*g3[i]
                    +beta4*g4[i]+beta5*g5[i]
                    +alpha1*branch1[i]+alpha2*branch2[i]
                    +branch[branchno[i]]+parcel[parcelno[i]]
      }
   alpha1~ dnorm(0.0,1.0E-6)
   alpha2~ dnorm(0.0,1.0E-6)
   beta1 ~ dnorm(0.0,1.0E-6)
   beta2~ dnorm(0.0,1.0E-6)
   beta3 ~ dnorm(0.0,1.0E-6)
```

```
   beta4 ~ dnorm(0.0,1.0E-6)
   beta5~ dnorm(0.0,1.0E-6)
   tau2parcel<-1/tau2parcelrec
   tau2branch<-1/tau2branchrec
  }

Initial values

list(branch=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0),
     parcel=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
     beta1=0,beta2=0,beta3=0,beta4=0,mu=0,alpha1=0,alpha2=0,
     tau2branchrec=1,tau2parcelrec=1)
```

A.5 Exercises for Lecture IV

A.5.1 Exercise8.R

```
#This is a toy example illustrating how you can use a sample from N(0,1) and importance sampling



#generate sample from N(0,1)
sample=rnorm(10000)
#estimate mean of N(1,0.5)
mean(sample*dnorm(sample,1,sqrt(0.5))/dnorm(sample))
       /mean(dnorm(sample,1,sqrt(0.5))/dnorm(sample))

#Next we use a sample from BUGS (produced with the coda button) and try
#to see what happens with posterior mean for tau2branchrec if we decrease
# variance in gamma prior

tau2branchrec=read.table("tau2branchrec.out",header=F)
#skip first 1000 samples (burn-in)
tau2branchrec=tau2branchrec[1001:100000,]
#posterior mean for original gamma(0.001,0.001) prior
#(mean is 1=0.001/0.001 variance is 0.001/0.001^2=1000
mean(tau2branchrec[,2])
0.2254103

#posterior mean for gamma(0.01,0.01) prior (mean is 1=0.01 variance is 0.01/0.01^2=100
mean(tau2branchrec[,2]*dgamma(tau2branchrec[,2],shape=0.01,rate=0.01)
        /dgamma(tau2branchrec[,2],shape=0.001,rate=0.001))
        /mean(dgamma(tau2branchrec[,2],shape=0.01,rate=0.01)
        /dgamma(tau2branchrec[,2],shape=0.001,rate=0.001))
0.2255676

#mean not affected

#posterior mean of variance tau2branch=1/tau2branchrec
mean(1/tau2branchrec[,2])
4.915679
```

A.5.2 Cherriesbinary.odc

```
Model with a binary random branch effect

model
   {
     q ~ dunif(0.0,1.0)
     for (i in 1:60){
        branch[i]~dbin(q,1)
        }
     for( i in 1 : 953 ) {
        dummy[i]<-no[i]
        dummy2[i]<-parcelno[i]
        y[i] ~ dbin(p[i],1)
        logit(p[i]) <- beta1*g1[i]+ beta2*g2[i]+beta3*g3[i]+beta4*g4[i]
                      +beta5*g5[i]+ branch[branchno[i]]
        }

     beta1 ~ dnorm(0.0,1.0E-6)
     beta2 ~ dnorm(0.0,1.0E-6)
     beta3 ~ dnorm(0.0,1.0E-6)
     beta4 ~ dnorm(0.0,1.0E-6)
     beta5~ dnorm(0.0,1.0E-6)
     }

initial values

list(branch=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0),
     beta1=0,beta2=0,beta3=0,beta4=0,beta5=0,q=0.5)
```